I3000 User's Guide

Revision: V.1.00 Date: March 29, 2013

www.holtek.com

Contents

I3000 Usage 1

Start Interface 1

Main Interface 3

Area Functions in the Main Interface 4

Function Introductions 4

Using the I3000 for Program Operation 10

Software Development Kit (SDK) 12

Operating Introduction 12

Operating Flowchart 13

SDK API Introduction 14

I3000 Usage

Start Interface

After the I3000 installation is finished, double-click the icon on the desktop, then the following interface will appear:

Figure 1

"USB Connect" indicates whether the software has connected to the Bootloader, "Download Mode" indicates if it is ready for program operations. ✔ means yes, ✖ means no. Figure 1 indicates that it has been connected to the Bootloader and normal program operations are allowed, while Figure 2 indicates no.

Figure 2

Click the <Setting> button to select the data type. Click the white check boxes, ☑ means being selected, ☐ means no.

Figure 3

Click the <Help> button to view the software information, clicking the following link will open this User's Guide.

Figure 4

Move the mouse to the button beside "Exit", when the button turns red as following shows, a single click on it will exit the software.

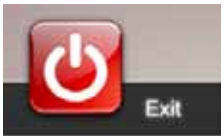


Figure 5

Click the <Open> button, an open file dialogue box appears. Only those files with an mtp suffix can be selected from any location on the computer.

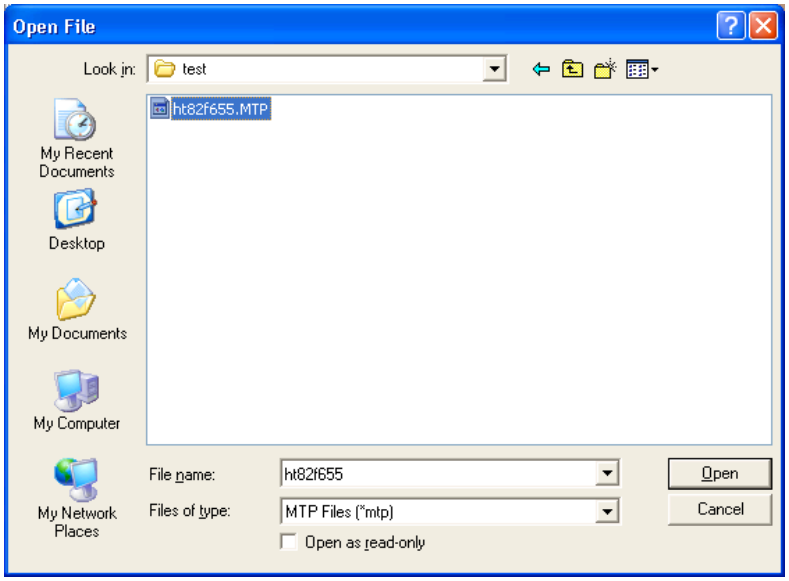


Figure 6

Double-click the required file or single-click the file and click the <Open> button to open the file and enter the main interface, as Figure 6.

Main Interface

The main Interface is mainly composed of the title bar, data area, function button area and status area, as Figure 7.

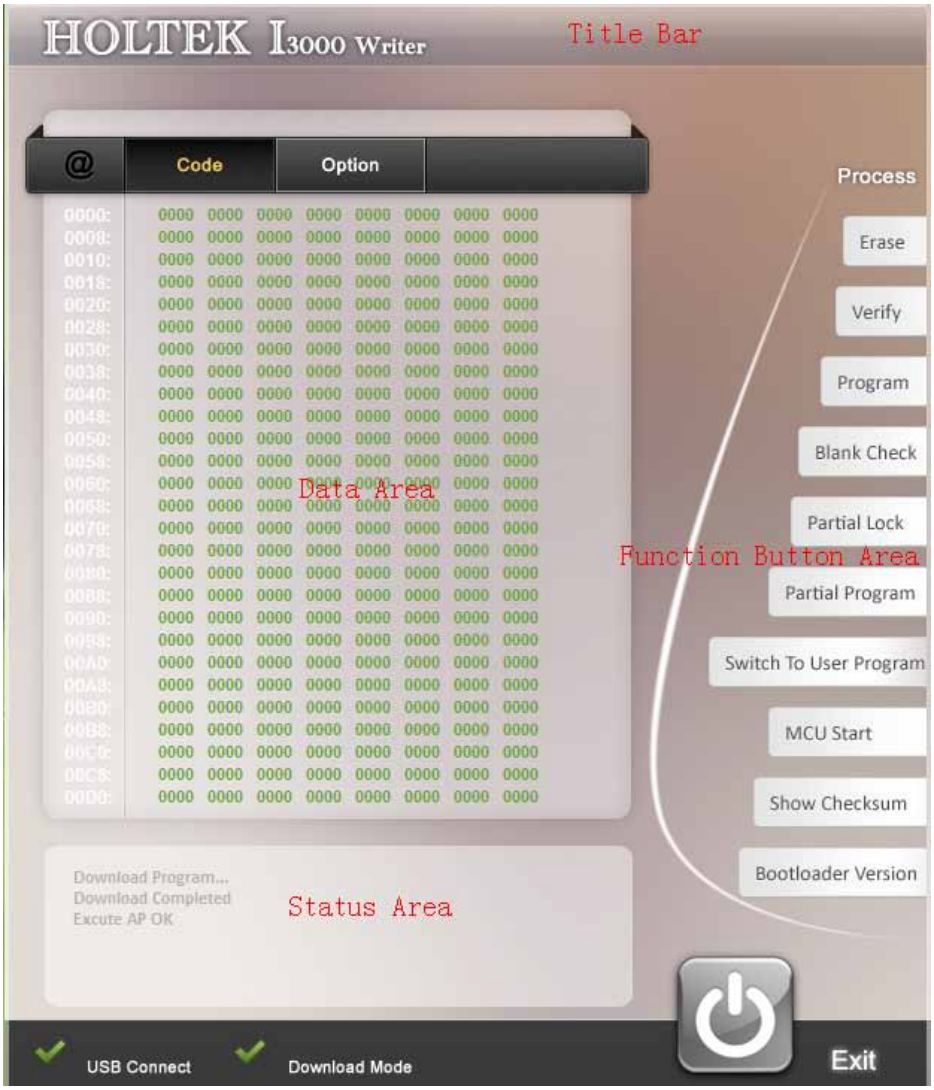


Figure 7

Move the mouse near to the title bar, a tool bar will be shown as below.

Figure 8

Using this tool bar users can open new files, reset data type or view the software information.

Area Functions in the Main Interface

The data area includes code and option two sections, which can switch to each other using a single click. If not all the contents are shown, move the mouse to the right border then a scroll bar appears, with which users can view the rest contents.

The function button area includes all the available operations.

The status area is mainly used to display the operation results.

Function Introductions

Erase

This instruction erases the code or option information programmed to the IC.

Click the <Erase> button, a dialogue box as Figure 9 will be shown.

Figure 9

By inputting a hexadecimal number in the Start Page or End Page text box, users can select the page where this address exists. Press <Click> for drop-down list to check the selected pages. Users can also select pages directly in the drop-down list, if the selected pages are consecutive, then the corresponding start address and end address will be displayed in the Start Page and End Page text box respectively, as Figure 10.

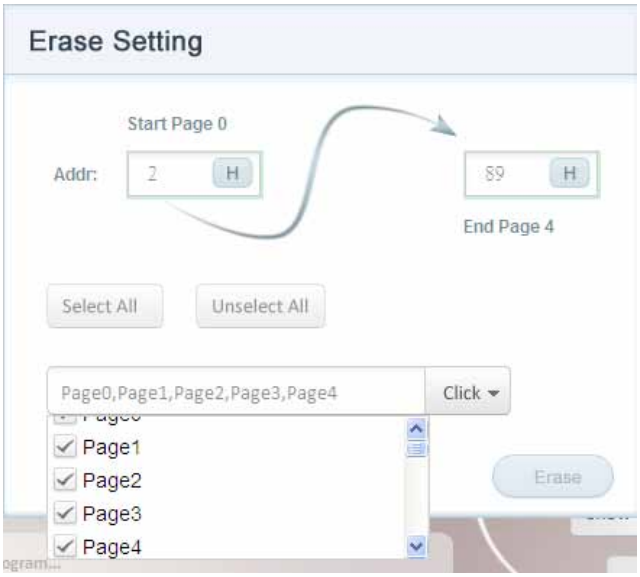


Figure 10

Clicking the <Select All> button will select all pages, clicking the <Unselect All> button will cancel all the selected pages.

Clicking the <Cancel> button will cancel erase operation.

Clicking the <Erase> button will start the erase operation. If no page is selected or the input address is not valid for a correct page range, a warning as shown below appears.

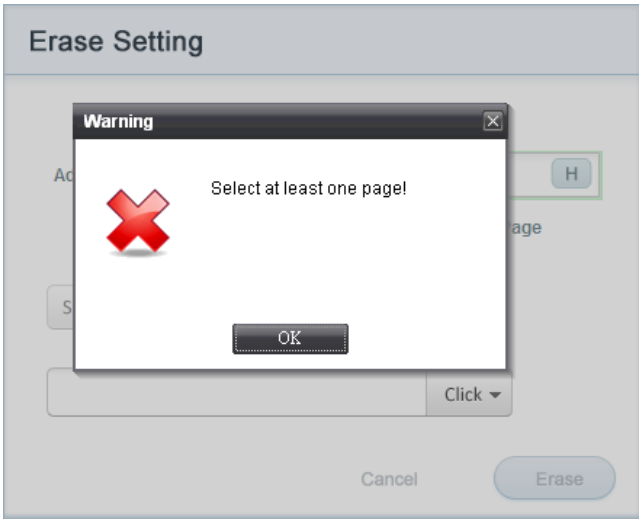


Figure 11

Otherwise a normal erase operation starts and a progress bar appears. If only the code check box in the Setting menu is selected, when the progress bar displays 100%, the status area will display "Erase program OK", and a <Close> button will appear being available to close the progress bar, as Figure 12.

Figure 12

Verify

This instruction verifies whether the code or option information has been programmed to the IC successfully.

Click the <Verify> button, a Verify Setting dialogue box will be shown.

Figure 13

This dialogue box is operated as same as the Erase Setting dialogue box.

Program

This instruction programs code or option information to the IC.

Click the <Program> button to start the program operation. This instruction is executed according to the selection in the Setting menu.

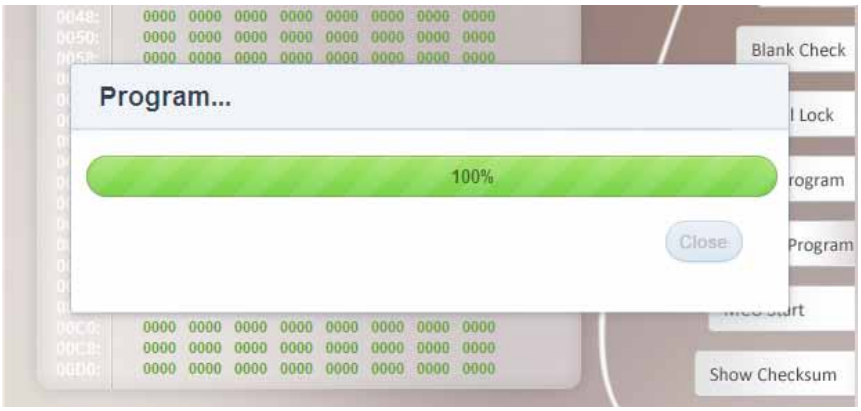


Figure 14

Note: If an error warning appears during the program operation, please execute an erase operation then try again.

Blank Check

This instruction checks if the IC has been previously programmed to.

Click the <Blank Check> button, if the IC has been programmed to, an error warning of "Not Blank" will be displayed, and then a <Close> button appears, as Figure 15. Otherwise the <Close> button appears only after the progress bar displays 100%, as Figure 16.

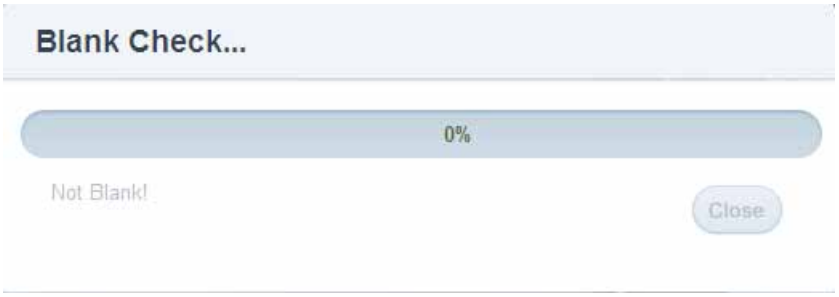


Figure 15

Figure 16

Partial Lock

This instruction locks the selected code part. After being locked, when an erase operation is executing, the selected code can not be read or erased.

Click the <Partial Lock> button, a dialogue box as shown below appears.

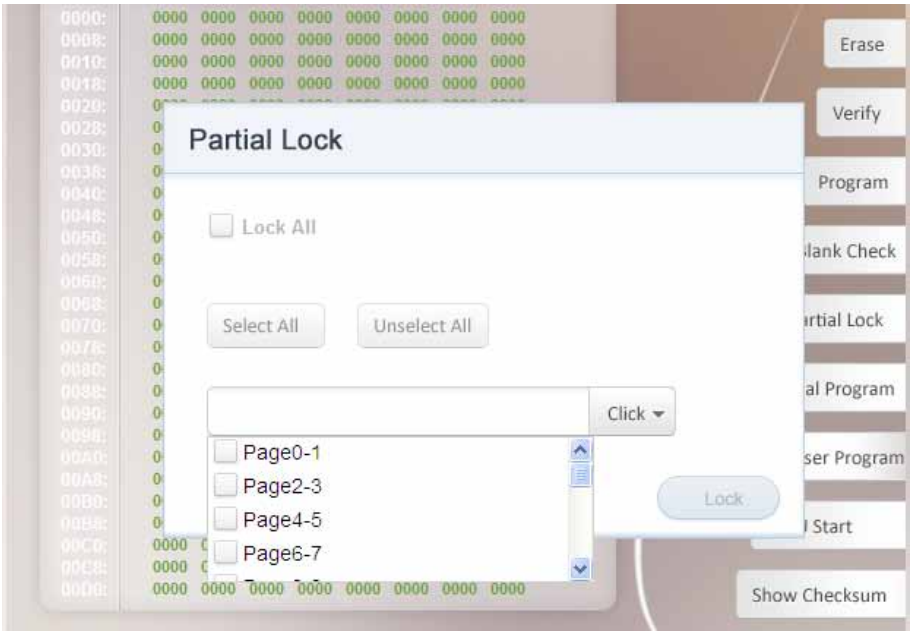


Figure 17

Press <Click> to view the drop-down list, clicking the white check box will select the corresponding page. Clicking the <Select All> or <Unselect All> button will select all pages or cancel all the selected pages. After select the required pages, click the <Lock> button to start the lock operation.

Partial Program

This instruction works as same as the Program instruction except the difference that this instruction only programs the specified pages.

Click the <Partial Program> button, the following dialogue box appears.

Figure 18

Input a hexadecimal number in the Start Page and End Page text boxes respectively, and make sure that the latter value is bigger than or equal to the former value. Click the <Program> button to program the selected pages.

Switch to User Program

This instruction simulates a USB plugging in and out process.

Click the <Switch to user program> button, the following dialogue box appears.

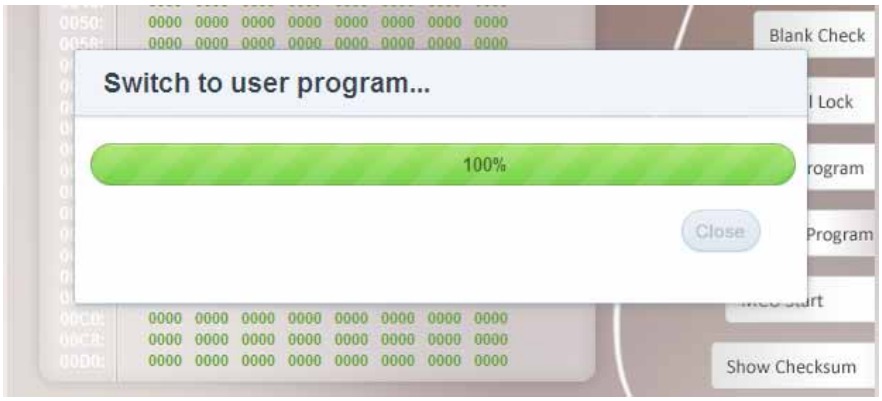


Figure 19

MCU Start

Only after Verify the IC has been successfully programmed to, can this instruction be used to select the MCU start mode.

Click the <MCU Start> button, a dialogue box as Figure 20 appears.

MCU Start
MCU starts at the bootloader
MCU starts at the bootloader then executes the user firmware
MCU starts at the user firmware
Cancel
OK

Figure 20

MCU Starts at the bootloader – starts at the bootloader;

MCU starts at the bootloader then executes the user firmware – starts at the bootloader before executing the user firmware;

MCU starts at the user firmware – starts at the user firmware.

Show Checksum

Selecting "Code" means to check Code ROM only.

Selecting "Code+Option" means to check Code ROM and Option ROM.

Click the <Show Checksum> button, a dialogue box as Figure 21 appears.

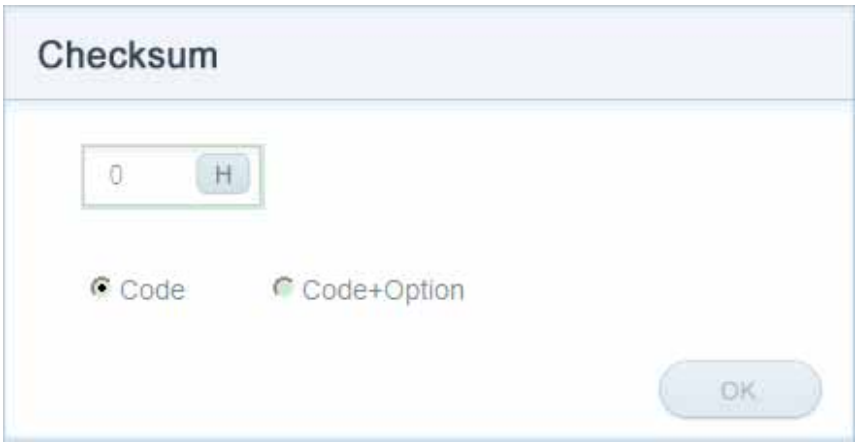


Figure 21

Select "Code" or "Code+Option" check box, the text box will display the corresponding Checksum value.

Bootloader Version

This instruction is used to obtain the Bootloader version.

Click the <Bootloader Version> button, the Bootloader version information will be displayed in the status area, as Figure 22.

Figure 22

Using the I3000 for Program Operation

- Step 1

  Click the <Open> button in the I3000 start interface or main interface to open the required MTP file, as Figure 23 and Figure 24.

Figure 23

Figure 24

- Step 2

  Make sure that it has been connected to the bootloader, click the <Program> button to begin the program operation, as figure 25. Then a process bar appears, as Figure 26.

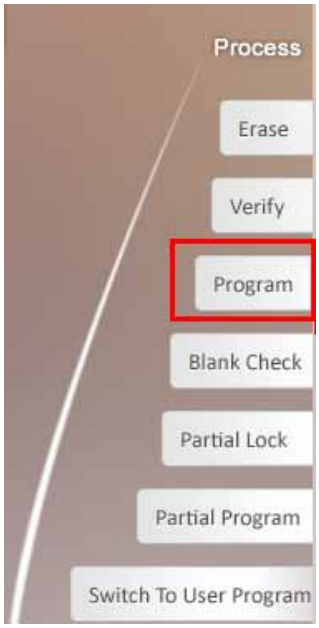


Figure 25

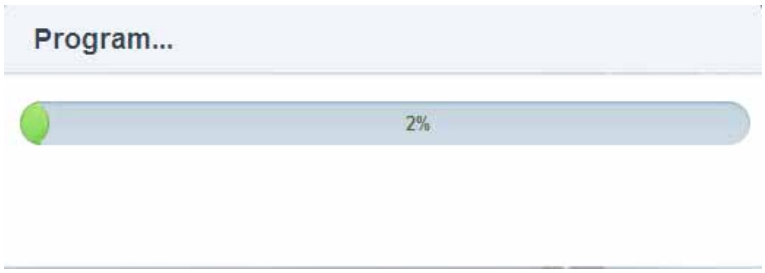


Figure 26

When the program operation is completed, the process bar will display 100%, and a <Close> button will be shown, as Figure 27.

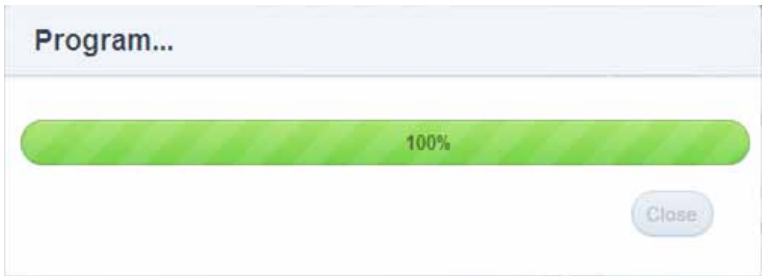


Figure 27

Click the <Close> button to close the process bar thus finishing the program operation.

Software Development Kit (SDK)

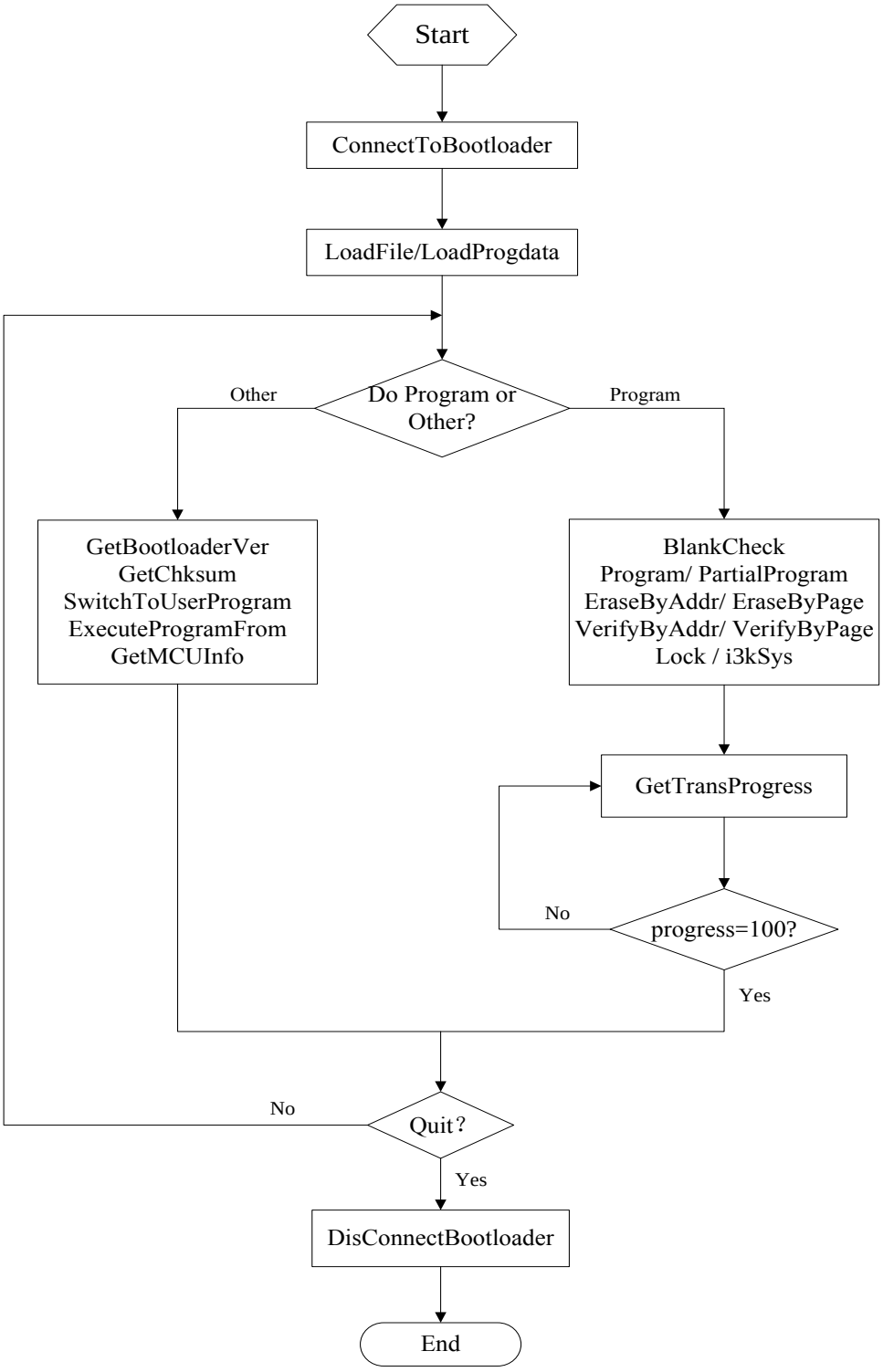
After the I3000 installation is completed, the installation directory includes an ISPDLL folder, which contains two I3000 SDK versions, the x86 used for 32-bit program and the x64 used for 64-bit program. Both two SDK versions contain all I3000 functions, users can choose any one version according to the requirement.

Operating Introduction

There are two methods to access the DLL file, the explicit linking and implicit linking. If users use the implicit linking method, an ISPDLL.h file must be included in the project and link the library file ISPDLL.lib. To avoid a conflict, users should not define WRITERCMD_API in the user application program.

When the application program is running, the ISPDLL.dll file, HIDDLL.dll file and EFORMAT.ini file must be located in the application program folder. The ISPDLL.dll file includes the required API, and the HIDDLL.dll file provides the underlying interface for the ISPDLL.dll file. When the application program is running in the Win7, to make sure the ISPDLL.dll file can be properly called, the operating system must run the application program as an administrator.

Operating Flowchart



SDK API Introduction

The ISPDLL.dll file includes 19 functions, which will be introduced below.

LoadFile

The function is used to load MTP file, and return the pointers of the Code buffer, Option buffer and Data buffer (if there is any data) and their buffer sizes by Byte.

int LoadFile(LPCTSTR szMtpPath, PBYTE& pProgramBuf, WORD& wProgramSize, PBYTE& pOptionBuf, WORD& wOptionSize, PBYTE& pDataBuf, WORD& wDataSize);

- **Parameters**

[in]szMtpPath

This is the full path of the MTP file to be loaded. If this parameter is input incorrectly, the function will fail.

[out] pProgramBuf

It points to the Code buffer.

[out] wProgramSize

It indicates the Code buffer size by byte.

[out] pOptionBuf

It points to the Option buffer.

[out] wOptionSize

It indicates the Option buffer size by byte.

[out] pDataBuf

It points to the Data buffer, if there is no data, it will return NULL.

[out] wDataSize

It indicates the Data buffer size by byte. If there is no data, it will return 0.

- **Return Value**

If the function fails, the return value is -1; otherwise 0.

- **Note**

This function needs to load an EFORMAT.ini configuration file internally. If there is no EFORMAT.ini file in the application directory, the function will fail. Either the LoadFile or LoadProgdata function must be called before a program type function, such as BlankCheck, Program, PartialProgram, EraseByAddr, EraseByPage, VerifyByAddr, VerifyByPage, Lock or GetChksum.

- **Sample Code**

```
CString strPath = _T("C:\\ht82f655.MTP ");
PBYTE pProgram = NULL, pOption = NULL, pData = NULL;
PBYTE pProgramBuf = NULL, pOptionBuf = NULL, pDataBuf = NULL;
WORD wProgramSize = 0, wOptionSize = 0, wDataSize = 0;
if(LoadFile(strPath, pProgram, wProgramSize, pOption, wOptionSize, pData, wDataSize)!= -1)
{
     pProgramBuf = new BYTE[wProgramSize];
     memcpy(pProgramBuf, pProgram, wProgramSize);
     pOptionBuf = new BYTE[wOptionSize];
     memcpy(pOptionBuf, pOption, wOptionSize);
     if (pData != NULL)
     {
              pDataBuf = new BYTE[wDataSize];
              memcpy(pDataBuf, pData, wDataSize);
     }
}

//Execute other operations……

//Release space when the application program is over
if(pProgramBuf != NULL)
{
     delete []pProgramBuf;
     pProgramBuf = NULL;
}
if(pOptionBuf != NULL)
{
     delete []pOptionBuf;
     pOptionBuf = NULL;
}
if(pDataBuf != NULL)
{
     delete []pDataBuf;
     pDataBuf = NULL;
}
```

LoadProgdata

Being same to the LoadFile function, the function uses the loaded MTP file to return the pointers of the Code buffer, Option buffer and Data buffer (if there is any data) and their buffer sizes by Byte.

int LoadProgdata(PBYTE pMtpBuf, DWORD dwMtpSize, PBYTE& pProgramBuf, WORD& wProgramSize, PBYTE& pOptionBuf, WORD& wOptionSize, PBYTE& pDataBuf, WORD& wDataSize);

- **Parameters**

[in] pMtpBuf

It points to the MTP buffer and can not be NULL, or the function will fail.

[in] dwMtpSize

It indicates the MTP buffer size and can not be 0, or the function will fail.

[out] pProgramBuf

It points to the Code buffer.

[out] wProgramSize

It indicates the Code buffer size by byte.

[out] pOptionBuf

It points to the Option buffer.

[out] wOptionSize

It indicates the Option buffer size by byte.

[out] pDataBuf

It points to the Data buffer. If there is no data, it returns NULL.

[out] wDataSize

It indicates the Data buffer size by byte. If there is no data, it returns 0.

- **Return Values**

If the function fails, the return value is -1; otherwise 0.

- **Note**

This function needs to load an EFORMAT.ini configuration file internally. If there is no EFORMAT.ini file in the application directory, the function will fail. Either the LoadFile or LoadProgdata function must be called before a program type function, such as BlankCheck, Program, PartialProgram, EraseByAddr, EraseByPage, VerifyByAddr, VerifyByPage, Lock or GetChksum.

- **Sample Code**

```
CString strPath = _T("C:\\ht82f655.MTP ");
PBYTE  pMtpBuf = NULL; //MTP file buffer
DWORD dwMtpFileSize = 0; //File size, initialized as 0
ReadFileToBuffer(strPath, pMtpBuf, dwMtpFileSize); //Read file to buffer
     PBYTE pProgram = NULL, pOption = NULL, pData = NULL;
     PBYTE pProgramBuf = NULL, pOptionBuf = NULL, pDataBuf = NULL;
     WORD wProgramSize = 0, wOptionSize = 0, wDataSize = 0;

if(LoadProgdata(pMtpBuf, dwMtpFileSize, pProgram, wProgramSize, pOption, wOptionSize,
pData, wDataSize)!= -1)
{
     pProgramBuf = new BYTE[wProgramSize];
     memcpy(pProgramBuf, pProgram, wProgramSize);
     pOptionBuf = new BYTE[wOptionSize];
     memcpy(pOptionBuf, pOption, wOptionSize);
     if (pData != NULL)
     {
              pDataBuf = new BYTE[wDataSize];
              memcpy(pDataBuf, pData, wDataSize);
     }
}
//Release file buffer
if(pMtpBuf != NULL)
{
     delete []pMtpBuf;
     pMtpBuf = NULL;
}

// Execute other operations……

//Release space when the application program is over
if(pProgramBuf != NULL)
{
     delete []pProgramBuf;
     pProgramBuf = NULL;
}
if(pOptionBuf != NULL)
{
     delete []pOptionBuf;
     pOptionBuf = NULL;
}
if(pDataBuf != NULL)
{
     delete []pDataBuf;
     pDataBuf = NULL;
}
```

GetChksum

The function is used to get the checksum.

int GetChksum(BYTE calcMode);

- **Parameter**

  [in] calcMode

  It specifies the check range. 0 stands for "Code", 1 stands for "Code + Option", 2 stands for "Code + Option + Data". For other values, the function will fail.

- **Return Value**

  If the function fails, the return value is -1; otherwise, the return value is a valid checksum.

- **Note**

  After the function is called, the program must first check if the return value is -1, if no, then convert the checksum to a WORD or BYTE type number according to the requirement.

- **Sample Code**

```
//Get Code buffer checksum
int iRet = GetChksum(0);
if(iRet != -1)
{
    //Display the checksum
    TCHAR buf[10];
    _itot(iRet, buf, 16); //Convert the checksum to a hexadecimal number
    MessageBox(buf);
}
```

BlankCheck

The function checks whether the specified type of data in the MCU is blank.

int BlankCheck(BYTE ucType);

- **Parameter**

  [in] ucType

  It stands for the data type to be checked. 0 stands for Code, 1 stands for Option. For other values, the function will fail.

- **Return Value**

  If the function fails, the return value is -1.

  If the function succeeds, a return value of 1 stands for non-blank, 0 stands for blank.

- **Note**

  The size of the Code or Option buffer is read from the EFORMAT.ini using the MTP file. Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Code buffer BlankCheck, the following code runs in a separate thread
int iRet = BlankCheck(0);
if(0 == iRet)
    MessageBox(_T("Blank!"));
else if(1 == iRet)
    MessageBox(_T("Non-blank!"));
else
    MessageBox(_T("BankCheck Failed!"));
```

Program

The function programs the specified type of data to the IC.

int Program(BYTE ucType);

- **Parameter**

[in] ucType

It indicates the data type to be programmed. 0 stands for Code, 1 stands for Option. For other values, the function will fail.

- **Return Value**

If the function fails, the return value is -1; otherwise 0.

- **Note**

Make sure that the LoadFile or LoadProdata function has already been called successfully before this function.

- **Sample Code**

```
//Programming code, runs in a separate thread
int iRet = Program(0);
if(iRet != -1)
    MessageBox(_T("Program Success!"));
else
    MessageBox(_T("Program Failed!"));
```

PartialProgram

The function programs the specified range of data to the IC.

int PartialProgram(WORD wOffset, WORD wordLen);

- **Parameters**

[in] wOffset

It indicates the starting address of the specified data range, one word as a unit. If this address exceeds the maximum address of the MCU available space, the function will fail.

[in] wordLen

It indicates the total length of data to be programmed, one word as a unit. If this length is greater than the MCU available space size, the excess part will be truncated.

- **Return Values**

If the function fails, the return value is -1; otherwise 0.

- **Note**

This function is only available for Code programming. The Option programming does not support the function.

This function converts the specified address range into pages. If the address range indicated by wOffset/ wordLen is not enough for integer pages, this function will internally align it into integer pages. If the address range exceeds the available space, the excess part will be truncated.

Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Program 32 words starting from address 0, the following code runs in a
//separate thread
int iRet = PartialProgram(0, 32);
if(iRet != -1)
    pDlg->MessageBox(_T("PartialProgram Success!"));
else
    pDlg->MessageBox(_T("PartialProgram Failed!"));
```

EraseByAddr

The function erases the specified data by address.

int EraseByAddr(WORD wOffset, WORD wordLen, BYTE ucType);

- **Parameters**

[in] wOffset

It is the starting address of the address range to be erased, one word as a unit. If this parameter is greater than the maximum address of the MCU available space, the function will fail.

[in] wordLen

It is the total length of the address range to be erased, one word as a unit. When ucType = 0, wordLen can not be 0, or the function will fail. If this parameter exceeds the MCU available space size, the excess part will be truncated.

[in] ucType

It indicates the programmed data type. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType=1, the above two parameters will be ignored.

- **Return Values**

If the function fails, the return value is -1; otherwise 0.

- **Note**

When ucType=0, this function will convert the specified address range into pages. If the address range indicated by wOffset/wordLen is not enough for integer pages, this function will internally align it into integer pages. If the address range exceeds the available space, the excess part will be truncated.

Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Erase 32 words starting from address 0 of the code buffer, the following
//code runs in a separated thread
int iRet = EraseByAddr(0, 32, 0);
if(iRet != -1)
    MessageBox(_T("Erase Success!"));
else
    MessageBox(_T("Erase Failed!"));
```

EraseByPage

The function erases the specified data by page.

int EraseByPage(WORD wStartPage, WORD wCnt, BYTE ucType);

- **Parameters**

[in] wStartPage

It specifies the zero-based index of starting page to be erased. If it is greater than the maximum page number in the MCU, the function will fail.

[in] wCnt

It is the number of pages to be erased. When ucType = 0, this parameter can not be 0, or the function will fail. If this parameter is greater than the maximum number of pages in the MCU, the excess part will be truncated.

[in] ucType

It indicates the data type to be erased. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType = 1, the above two parameters will be ignored.

- **Return Value**

If the function fails, the return value is -1; otherwise 0.

- **Note**

Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Erase 100 pages starting from page 0 of the code buffer, the following code
//runs in a separated thread
int iRet = EraseByPage(0, 100, 0);
if(iRet != -1)
    MessageBox(_T("Erase Success!"));
else
    MessageBox(_T("Erase Failed!"));
```

VerifyByAddr

The function verifies whether the data of the specified addresses in the MCU match the programmed data.

int VerifyByAddr(WORD wOffset, WORD wordLen, BYTE ucType);

- **Parameters**

[in] wOffset

It is the starting address of the specified address range to be verified, one word as a unit. If this parameter is greater than the maximum address of the MCU available space, the function will fail.

[in] wordLen

It is the total length of the addresses to be verified, one word as a unit. When ucType = 0, wordLen can not be 0, or the function will fail. If this parameter exceeds the MCU available space size, the excess part will be truncated.

[in] ucType

It is the data type to be verified. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType = 1, the above two parameters will be ignored.

- **Return Value**

If the function fails, the return value is -1.

If the function succeeds, a return value of 0 indicates that all data matched, 1 indicates unmatched.

- **Note**

When ucType = 0, this function will convert the specified address range into pages. If the address range indicated by wOffset/wordLen is not enough for integer pages, this function will internally align it into integer pages. If the address range exceeds the available space, the excess part will be truncated.

Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Verify 64 words starting from address 0 of the code buffer, the following
//code runs in a separated thread
int iRet = VerifyByAddr(0, 64, 0);
if(0 == iRet)
    MessageBox(_T("Data Matched!"));
else if(1 == iRet)
    MessageBox(_T("Data Unmatched!"));
else
    MessageBox(_T("VerifyByAddr() function call is failed!"));
```

VerifyByPage

The function verifies whether the data of the specified pages in the MCU match the programmed data.

int VerifyByPage(WORD wStartPage, WORD wCnt, BYTE ucType);

- **Parameters**

  [in] wStartPage

  It specifies the zero-based index of starting page to be erased. If it is greater than the maximum page number in the MCU, the function will fail.

  [in] wCnt

  It is the number of pages to be verified. When ucType = 0, this parameter can not be 0, or the function will fail. If this parameter is greater than the maximum number of pages in the MCU, the excess part will be truncated.

  [in] ucType

  It indicates the data type to be verified. 0 stands for Code, 1 stands for Option. For ther values, the function will fail. When ucType = 1, the above two parameters will be ignored.

- **Return Values**

  If the function fails, the return value is -1.

  If the function succeeds, a return value of 0 indicates that all data matched, 1 indicates unmatched.

- **Note**

  Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Verify 100 pages starting from page 0 of the code buffer, the following code
//runs in a separated thread
int iRet = VerifyByPage(0, 100, 0);
if(0 == iRet)
    MessageBox(_T("Data Matched!"));
else if(1 == iRet)
    MessageBox(_T("Data Unmatched!"));
else
    MessageBox(_T("VerifyByPage() function call is failed!"));
```

Lock

The function locks pages where the specified addresses exit.

int Lock(WORD wOffset, WORD wordLen);

- **Parameters**

[in] wOffset

It is the starting address of the specified addresses to be locked, one word as a unit. If this parameter is greater than the maximum address of the MCU available space, the function will return 1.

[in] wordLen

It is the total length of addresses to be locked, one word as a unit. If this parameter is greater than the MCU available space size, the function will return 1.

- **Return Values**

If the function fails, the return value is -1.

If the function succeeds, a return value of 1 indicates wrong pages, 0 indicates a successful lock.

- **Note**

This function converts the address range specified by wOffset/wordLen into pages, 256 words as one page. If the specified address range is not enough for integer pages, this function will internally align it into integer pages. The Lock function usually locks two pages once, if the required pages number is odd, the number of pages actually being locked may be even.

If the address range specified by wOffset/wordLen equals to the MCU Code buffer size, then the whole Code buffer will be locked. Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Lock 320 words starting from address 0, the following code runs in a
//separated thread
int iRet = Lock(0, 320);
if(0 == iRet)
    pDlg->MessageBox(_T("Lock Success!"));
else if(1 == iRet)
    MessageBox(_T("Wrong Pages!"));
else
    MessageBox(_T("Lock failed!"));
```

GetTransProgress

The function gets the current programming progress.

int GetTransProgress();

- **Return Value**

The function returns the current completion ratio.

- **Note**

After a program type function, such as BlankCheck, Program, PartialProgram, EraseByAddr, EraseByPage, VerifyByAddr, VerifyByPage or Lock, is called, this function can be called to get the related programming progress, a completion ratio of 100 indicates that the program operation is finished.

- **Sample Code**

```
int nProgress = GetTransProgress();
CString str;
str.Format(_T("%d%%"), nProgress);
    MessageBox(str);
```

GetBootloaderVer

The function is used to get Bootloader version.

BYTE GetBootloaderVer();

- **Return Value**

After the function is called, the Bootloader version is returned.

- **Sample Code**

```
BYTE version = GetBootloaderVer();
CString str;
str.Format(_T("Bootloader version: %d"), version);
    MessageBox(str);
```

ConnectToBootloader

The function is used to connect to Bootloader.

int ConnectToBootloader();

- **Return Values**

If the function fails, the return value is -1; otherwise 0.

- **Sample Code**

```
if(ConnectToBootloader() != -1)
    MessageBox(_T("Connection Successful!"));
else
    MessageBox(_T("Connection Failed!"));
```

DisConnectBootloader

The function is used to disconnect to Bootloader.

int DisConnectBootloader();

- **Return Value**

If the function fails, the return value is -1; otherwise 0.

- **Sample Code**

```
if(DisConnectBootloader() != -1)
    MessageBox(_T("Disconnection Sucessful!"));
else
    MessageBox(_T("Disconnection Failed!"));
```

SwitchToUserProgram

The function is used to switch to user program.

int SwitchToUserProgram();

- **Return Value**

If the function fails, the return value is -1; otherwise 0.

- **Sample Code**

```
//The following code runs in a separated thread
int iRet = SwitchToUserProgram();
if(iRet!= -1)
    MessageBox(_T("Switching Successful!"));
else
    MessageBox(_T("Switching Failed!"));
```

ExecuteProgramFrom

The function is used to setup the program executing mode.

int ExecuteProgramFrom(BYTE byMode);

- **Parameter**

[in] byMode

It indicates the program executing mode.
0: MCU Starts at the bootloader;
1: MCU starts at the bootloader then executes the user firmware;
2: MCU starts at the user firmware;
For other values, the function will fail.

- **Return Value**

If the function fails, the return value is -1; otherwise 0.

- **Sample Code**

```
//Select Mode 0, the following code runs in a separated thread
int iRet = ExecuteProgramFrom(0);
if(iRet != -1)
    pDlg->MessageBox(_T("Setup Successful!"));
else
    pDlg->MessageBox(_T("Setup Failed!"));
```

GetMCUInfo

The function is used to get MCU page size or the maximum number of pages in Code buffer.

WORD GetMCUInfo(BYTE byType);

- **Parameter**

[in] byType

It indicates the information type. When byType = 1, it means to get the page size, one byte as a unit. When byType = 2, it means to get the maximum number of pages in Code buffer. For other values, the function will return 0.

- **Return Value**

Normally, the function will return the page size or the maximum number of pages in Code buffer. If the return value is 0, it means that the above parameter is invalid.

- **Note**

Make sure that the LoadFile or LoadProgdata function has already been called successfully before this function.

- **Sample Code**

```
//Get the page size
WORD wResult = GetMCUInfo(1);
if(wResult > 0)
{
    CString str;
    str.Format(_T("page size:%d bytes"), wResult);
    MessageBox(str);
}
```

i3kSys

The function reads a given size of data from MCU Code buffer and compares them with the specified data.

BOOL i3kSys(WORD wOffset, WORD wSize, BYTE* buffer)

- **Parameter**

[in] wOffset

It is the starting address of the MCU Code data to be compared, one word as a unit.

[in] wSize

It is the size of the MCU Code data to be compared, one byte as a unit. If this parameter is 0 or not integer multiple of the page size, the function will return FALSE.

[in] buffer

It points to the specified data buffer. If buffer = NULL, the function will return FALSE.

- **Return Values**

If the data read from the Code buffer match the specified data, TRUE is returned, otherwise FALSE is returned. In addition, if any one of the above parameters is invalid, the function will also return FALSE.

- **Sample Code**

```
//First to get the pointer of the specified buffer to be compared
//Then call GetMCUInfo(1)
//Compare n pages starting from address 0 of the Code space with the specified
//buffer (here n is a integer number except 0)
if(i3kSys(0, GetMCUInfo(1)*n, buffer) != FALSE)
    MessageBox(_T("TRUE"));
else
    MessageBox(_T("FALSE"));
```

Holtek Semiconductor Inc. (Headquarters)
No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

Holtek Semiconductor Inc. (Taipei Sales Office)
4F-2, No. 3-2, YuanQu St., Nankang Software Park, Taipei 115, Taiwan
Tel: 886-2-2655-7070
Fax: 886-2-2655-7373
Fax: 886-2-2655-7383 (International sales hotline)

Holtek Semiconductor (China) Inc.
Building No.10, Xinzhu Court, (No.1 Headquarters), 4 Cuizhu Road, Songshan Lake, Dongguan, China 523808
Tel: 86-769-2626-1300
Fax: 86-769-2626-1311

Holtek Semiconductor (USA), Inc. (North America Sales Office)
46729 Fremont Blvd., Fremont, CA 94538, USA
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com

Copyright© 2013 by HOLTEK SEMICONDUCTOR INC.
The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at http://www.holtek.com.tw.